SPECIAL FEATURE

This new 32-bit microprocessor provides high performance, instruction set extensibility, and compatibility with existing M68000 family software.

The Motorola MC68020

Doug MacGregor,
Dave Mothersole,
and Bill Moyer

Motorola, Inc.

The MC68020 represents the first successful extension of a 16-bit microprocessor into the 32-bit world—it provides a full 32-bit data and address bus as well as a 32-bit internal architecture. A natural and elegant 32-bit implementation of the M68000 architecture, it represents the state of the art in high-performance microprocessor design. It is designed to execute all user object code written for previous members of the M68000 family of processors and to execute those programs much faster. Because of the value of the large software base available for M68000 processors, the MC68020 maintains compatibility with previous members of the family.

In addition to performance and compatibility, there are several system enhancements associated with the new processor. It supports additional addressing modes, includes new instructions, extends all instructions to 32-bit operations, has a larger register set, and supports more data types. The MC68020 supports the virtual memory/virtual machine functionality of the MC68010, extends the 68010's support for modular programming, and provides a coprocessor interface for instruction set extension.

Although all MC68000 family members have 32-bit user registers, operands, and internal registers, the MC68020 adds full 32-bit data paths (internal and external), two 32-bit internal address paths, a 32-bit execution unit, three 32-bit arithmetic units, and an on-board instruction cache. The design goal was to provide a four- to sixfold performance improvement for the 16-MHz MC68020 over the 8-MHz MC68000. This performance increase is realized by a combination of the higher clock frequency at which the MC68020 operates, the 32-bit data bus width, the presence of the on-chip instruction cache, and the addressing mode enhancements.

Compatibility

One of the primary goals of the MC68020 was to maintain strict user object code compatibility with the M68000 family. This goal was easy to achieve because the M68000 architecture already was predominantly a 32-bit architecture. The only instruction set enhancements required to extend the family to a full 32-bit architecture entailed pro-

viding 32-bit displacements and supporting 32-bit operands for the few instructions—such as multiply and divide—that did not already support 32 bits. It is important to note that unlike many machines with less consistent or weaker architectures, the MC68020 requires no special modes or configurations to maintain compatibility with earlier family members.

The issue of compatibility is crucial to most users, as their most significant investment is in software. The MC68020 is completely user-object-code-compatible with the MC68010. That is not to imply that there are radical differences in the supervisor code; the changes have been restricted primarily to the exception-handling routines. Most of the differences are due to the MC68020's new stack formats and system debug features.

Because source code is often unavailable to the user for recompilation (assuming recompilation is an acceptable alternative), object code compatibility is becoming a major factor in processor selection. Compatibility is also significant for system designers, since their decisions are heavily influenced by the effort required for software development. By restricting required code changes to the exception handling routines, one can design a processor card to support the MC68020, alter the exception-handling routines, and begin operation in a system originally designed for the MC68010 or MC68000, all in a very short time. While this approach does not take full advantage of the features of the MC68020 other than the high performance of the micromachine, it allows a system to be bootstrapped from an existing MC68010 design and quickly brought to market. Such a system can later be retrofitted with the software and 32-bit buses needed to take advantage of all of the MC68020's enhancements. In this way, the MC68020 provides a 32-bit growth path for 8- and 16-bit systems based on the MC68000, MC68008, MC68010, and MC68012 processors.

Instruction set enhancements

The MC68020 instruction set includes and extends the MC68010 instruction set and adds new instructions, new data types, and new addressing modes.

Data types. The M68000 family supports data types of bits, bytes, words, long words, and binary-coded decimal (BCD). Variable-width bit fields and packed BCD, quad-word (8-byte), and variable-byte-length operands are four new data types supported by the MC68020. The bit field instructions operate on any bit field from 1 to 32 bits in length regardless of its location in register or memory. Quad-word operands are necessary for long divide and multiply instructions which handle the 64-bit dividend and product, respectively. Variable-byte-length operands are provided to support the coprocessor interface—coprocessors can define operand lengths suitable for the application.

Table 1.
New addressing modes provided by the MC68020.

DATA SPACE	PROGRAM SPACE
(bd,An,Xn.sz*scl)	(bd,PC,Xn.sz*scl)
([bd,An],Xn.sz*scl,od)	([bd,PC],Xn.sz*scl,od)
([bd,An,Xn.sz*scl],od)	([bd,PC,Xn.sz*scl],od)

bd = BASE DISPLACEMENT (0, 16, 32-BITS)
An = ONE OF EIGHT 32-BIT ADDRESS REGISTERS
Xn = INDEX REGISTER, ONE OF 16 DATA/ADDRESS REGISTERS SIZED (sz) AND SCALED (scl)
PC = PROGRAM COUNTER
od = OUTER DISPLACEMENT (0, 16, 32-BITS)
[ ] = MEMORY INDIRECTION

Addressing modes. The M68000 family supports 14 addressing modes including register direct, register indirect (with predecrement, postincrement, displacement, and indexing), absolute, immediate, program-counter-relative, and implied effective. The MC68020 provides improved support for arrays and lists by expanding the indexed addressing modes to include memory indirection, full 32-bit displacements, sign extension of displacements and index operands, index scaling, preindexing and postindexing, and suppression of base register or index value or both. These extensions are available for both data and program address space accesses. Table 1 gives the format of the new addressing modes. This is not a complete description, for any one of the elements can be suppressed (using a value of 0) in the calculation. The index register can be either a data or an address register; it can be word- or long-word-sized, and it can be scaled by 1, 2, 4, or 8. Memory indirection is denoted by braces—"[ ]"—which indicate that the address within the braces is used to fetch a long-word address that is then used in an effective address calculation. There are two displacements available when memory indirection is used, although they can be suppressed if they are not needed. When full address generation capability is not needed, a brief format that allows for fast indexing, scaling, and an 8- or 16-bit displacement can be used.

These new modes are significant because they can be used in addressing high-level-language data structures. These modes not only improve execution performance but they also make compiler-generated code much more efficient, since they are able to more naturally express the structures associated with high-level languages. Later, we will provide an example of these modes, along with a detailed description of their implications for performance.

Instruction extensions. The instruction set of the M68000 family required only minor modification to make it consistently 32-bit. Table 2 details the additions to the M68000 instruction set that were involved in this extension. The multiply and divide instructions now operate on 32-bit operands. It is now possible to multiply two 32-bit operands to form either a 32- or 64-bit product. A 64/32-bit and a 32/32-bit divide operation is also available. The displacements associated with instructions were expanded from 16 to 32 bits. This is an important capability—it allows branch and link operations to take place without restrictions across the full four-gigabyte address space. This is a significant improvement for compiler support. The move control register (MOVEC) instruction has been expanded to accommodate the new

control registers in the MC68020, and the breakpoint (BKPT) instruction has been enhanced to provide more real-time program debug support.

New instructions. Over 20 new instructions have been added to provide new functionality (see Table 3). Although most of them will be utilized in general-purpose processing, some are more significant for special-purpose processing.

A large group of the new instructions provides manipulation of variable-width bit fields. Here, a bit field of variable length (up to 32 bits), either in a data register or spanning up to five bytes in memory, may be cleared, set, complemented, extracted, inserted, scanned, or tested. These instructions are efficient at manipulating packed data and for communications and graphics applications. One feature of these instructions is that they make it possible, when dealing with a bit field in memory, to specify any 1- to 32-bit field regardless of its orientation in memory. There are no restrictions as to how that field is aligned in memory. As cited above, if a bit field spans five bytes, the processor can internally recognize that condition and make the appropriate adjustments.

One of the major additions to the instruction set is the group of general coprocessor instructions which support the coprocessor interface. Since it is not possible to support all of the special-purpose computation needs of the users that are serviced by a general-purpose processor such as the MC68020, it was essential to develop a strategy to deal with nongeneral applications. To serve this need, the coprocessor interface was developed to provide a mechanism by which Motorola-defined VLSI special-purpose coprocessors (e.g., the floating-point coprocessor, the paged memory management unit), VLSI coprocessors designed by other organizations, and user-defined discrete coprocessors could be used to augment the capabilities of the MC68020 without encumbering the architecture. In a general-purpose architecture, highly specialized instructions and data types cannot be fully supported directly. Part of the power of the MC68000 architecture is due to its general nature. Rather than attempting to support a nearly unlimited set of special-purpose applications incompletely, the MC68020's designers extended the general nature of the processor to provide lower-level primitives which can be combined in a number of ways to support these special applications.

The coprocessor instructions supplant the F-line exception mechanism which allowed for customer emulation of these special functions by taking a trap (an exception) to an emulation routine. Seven coprocessor instructions provide a mechanism for communicating instruction information and data to the coprocessor. These are complemented by a set of 18 response primitives which the coprocessor can use to request processor activity on its behalf. Typically, the coprocessor receives instruction information from the processor as a result of the coprocessor instruction. The coprocessor evaluates the support activity required and requests those services by issuing primitive responses to the processor.

Modular programming is supported by the call module (CALLM) and the return from module (RTM) instructions. These instructions provide a method of subdividing the user state into 256 access levels (levels of privilege), and then define a protocol for supporting a transition between access levels. Two different varieties of these instructions support simple module calls, which involve no access-level changes, and complex module calls, which may involve access-level changes.

The bounds-checking instructions, CHK and CMP, found on earlier members of the family have been extended to allow specification of both upper and lower bounds. With the CHK2 and the CMP2 instructions, it is possible to test if a register is between an upper and a lower bound. The difference between the two instructions is that if the value is out of bounds, CHK2 causes an exception to occur in addition to setting the condition codes. The CMP2 instruction sets the condition codes only. This test is done for either signed or unsigned bounds. The processor automatically evaluates the relationship between the two bounds to determine which type of comparison is appropriate.

There are several new instructions which are used to provide system support. The compare and swap (CAS) and compare and swap 2 (CAS2) instructions allow for locked-bus manipulation of byte, word, or long-word data elements to support system queue and stack functions. The pack and unpack instructions provide the ability to easily convert from packed ASCII/EBCDIC data

Table 2.
MC68020 instruction enhancements.

INSTRUCTIONS	ENHANCEMENTS
MULS, MULU, DIVS, DIVU	OPERATIONS EXTENDED TO 32-BIT OPERANDS
Bcc, BRA, BSR, LINK	DISPLACEMENTS EXTENDED TO 32 BITS
MOVEC	NEW CONTROL REGISTERS MAY BE ACCESSED
BKPT	OPCODE SUBSTITUTION SUPPORTED

Table 3.
New instructions provided by the MC68020.

BFCHG	BIT FIELD CHANGE
BFCLR	BIT FIELD CLEAR
BFEXTS	BIT FIELD SIGNED EXTRACT
BFEXTU	BIT FIELD UNSIGNED EXTRACT
BFFFO	BIT FIELD FIND FIRST ONE SET
BFINS	BIT FIELD INSERT
BFSET	BIT FIELD SET
BFTST	BIT FIELD TEST
CALLM	CALL MODULE
CAS	COMPARE AND SWAP
CAS2	COMPARE AND SWAP (TWO-OPERAND)
CHK2	CHECK REGISTER AGAINST UPPER AND LOWER BOUNDS
CMP2	COMPARE REGISTER AGAINST UPPER AND LOWER BOUNDS
cpBcc	COPROCESSOR BRANCH ON COPROCESSOR CONDITION
cpDBcc	COPROCESSOR TEST CONDITION, DECREMENT, AND BRANCH
cpGEN	COPROCESSOR GENERAL FUNCTION
cpRESTORE	COPROCESSOR RESTORE INTERNAL STATE
cpSAVE	COPROCESSOR SAVE INTERNAL STATE
cpSETcc	COPROCESSOR SET ACCORDING TO COPROCESSOR CONDITION
cpTRAPcc	COPROCESSOR TRAP ON COPROCESSOR CONDITION
PACK	PACK BCD
RTM	RETURN FROM MODULE
UNPK	UNPACK BCD

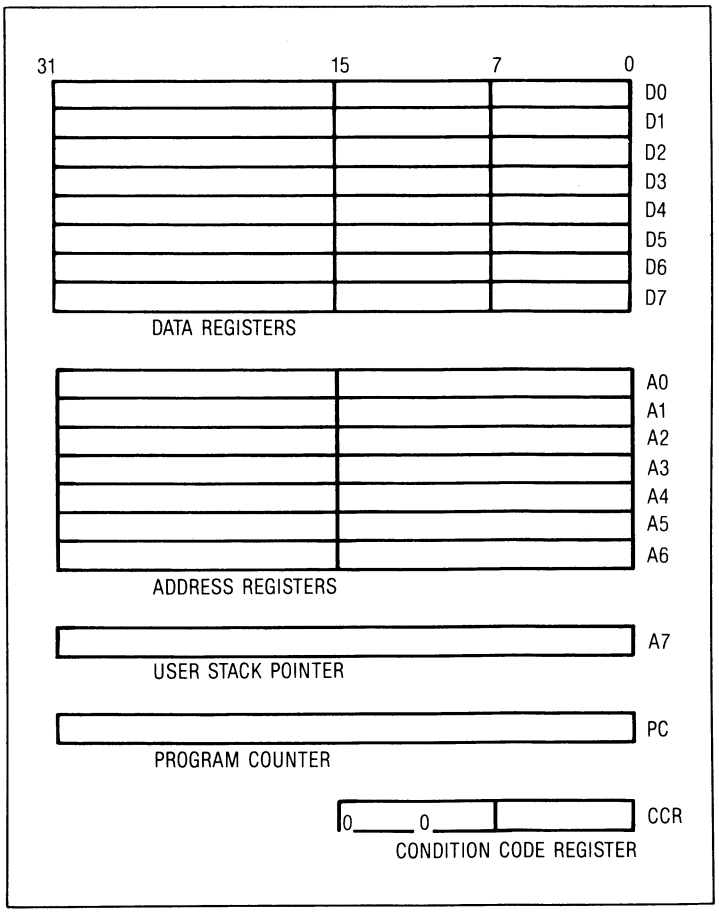

Figure 1. User programming model.

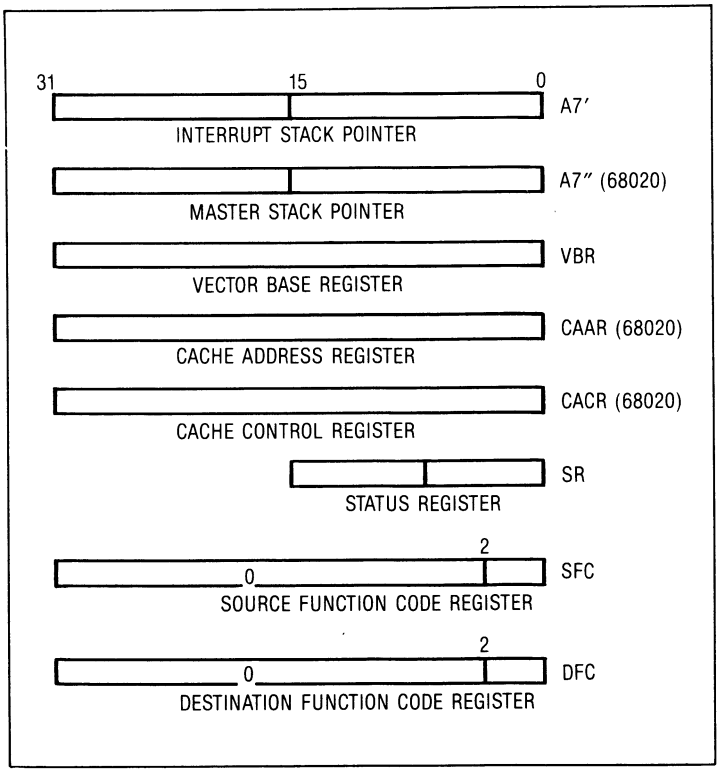

Figure 2. Supervisor programming model.

formats to BCD format, or back, for arithmetic operations. The breakpoint instruction provides system debug assistance to the user via the breakpoint-acknowledge mechanism. With this instruction, it is possible to breakpoint an instruction. When the processor attempts to execute the breakpointed instruction, a breakpoint-acknowledge cycle is executed and the proper opcode is fetched and substituted into the instruction pipe in place of the breakpoint, if appropriate, or a breakpoint routine is executed.

Programmer's model

The programmer's model that is presented to the user is shown in Figure 1. This model is identical to that of every other M68000 processor.

The supervisor programmer's model (Figure 2) contains some additions. There are two registers used to control the cache and an additional stack pointer for use with process-related exceptions. The cache control register (CACR) is used to control the cache and to access the status of the cache. The cache address register (CAAR) is used as a pointer into the cache for those control operations affecting only one entry in the cache. The use of the master stack pointer is described in greater detail below.

The MC68020 status register has two newly defined bits (see Figure 3). An additional bit, T0, when combined with the previous trace bit, now T1, allows for the definition of three trace states: no trace, trace on change of flow, and trace on instruction execution. The addition of the trace on change of flow allows traces to occur only after the execution of an instruction in which the program counter is not advanced sequentially (i.e., a branch, JMP, or RTS). The other addition is the master bit (M), which is used to specify when the processor is in the master state. This bit, when used in conjunction with the supervisor bit (S), indicates which register is to be accessed as the system stack pointer. This additional stack (if the M bit is set) is used only in the stacking of process-related exceptions. Compatibility is maintained, however, since at reset the M bit is cleared; with the M bit cleared, the behavior is identical to the other processors in the M68000 family.

The M68000 address space is partitioned into five address spaces: User data, user program, supervisor data, supervisor program, and CPU space. This partitioning has been refined and extended in the MC68020, as shown in Table 4. A user-reserved space dedicated to user applications has been defined. This space can only be accessed by the processor using the move space (MOVES) instruction.

In addition to this, the CPU space has been subdivided into 16 types of accesses. The type of CPU access is indicated by address bits 19-16 in combination with the CPU space function code (FC = 111). Table 5 indicates the different types of CPU accesses defined. All undefined spaces are reserved for future expansion. The definition of regions in the CPU space makes it possible to acknowledge breakpoints and interrupts, as well as to provide a means for communicating with coprocessors and

other special devices (i.e., memory management units), without dictating memory organization for user- and supervisor-related activity.

Coprocessor interface

The MC68020's coprocessor interface is a powerful and flexible mechanism capable of extending the MC68020 basic instruction set and supporting new data types. This interface is designed to support concurrent and nonconcurrent coprocessors and bus master and bus slave coprocessors without being encumbered by the specifics of an application. By providing this functionality, the interface offers a solution to the special processing problems that system designers face. With this mechanism, the system designer can use either a VLSI coprocessor to solve some of the more common special-purpose processing needs (i.e., floating-point computations) or a board-level design particular to some limited but very important application. The coprocessor interface is based on external bus cycles. CPU space reads and writes are used to transfer information between the main processor and the coprocessor. The coprocessor interface registers are memory-mapped into the CPU space. Coprocessor operation is controlled by sequences of writes and reads of these coprocessor control registers. The interface, though optimized for the MC68020, can also be emulated by older processors in the family, which can deal with the coprocessor as a peripheral. Finally, since we know that it is impossible to define an interface to solve every application problem, there are mechanisms available which allow for software emulation of undefined primitives—in this way, if a function is not currently defined, it can be either added later or at least emulated by the user.

Coprocessor instructions (F-line instructions) from the MC68020 are passed to one of eight possible coprocessors for evaluation. This decouples the main processor from the definition of a particular coprocessor instruction set and provides a great deal of generality. After evaluating the instruction, the coprocessor determines the services it requires from the main processor (if any) so that it can execute the instruction. These service requests are transmitted to the main processor in the form of primitives, which are then evaluated and executed by the main processor. There has been an effort to make these primitives very powerful so that there can be a minimum amount of overhead in the coprocessor interface. For example, the main processor can perform an effective address calculation and then either pass the evaluated effective address to the coprocessor or fetch a variable-length operand and pass that to the coprocessor. This is a complicated process, but by providing this service the main processor allows the coprocessor design to be much simpler.

The MC68020 supports five types of coprocessor instructions: general, branch, conditional, save state, and restore state. The general instruction (cpGEN) is used for data movement and manipulation. The branch instruction (cpBcc) allows branching based on a coprocessor condition. Both 16- and 32-bit branch displacements are supported. The conditional instructions (cpScc, cpTRAPcc, and cpDBcc) allow evaluation of one of 64 coprocessor conditions, and they conditionally set (cpScc), trap (cpTRAPcc), or decrement and branch (cpDBcc). The coprocessor is passed a condition for evaluation and then informs the main processor of the result. In addition to the general, branch, and conditional types, the save type

Table 4.
Function code definition.

FC2-0	SPACE
000	UNUSED (RESERVED)
001	USER DATA
010	USER PROGRAM
011	USER RESERVED
100	UNUSED (RESERVED)
101	SUPERVISOR DATA
110	SUPERVISOR PROGRAM
111	CPU SPACE

Table 5.
CPU space access types.

A19-16	TYPE OF ACCESS
0000	BREAKPOINT ACKNOWLEDGE
0001	MEMORY MANAGEMENT UNIT
0010	COPROCESSOR
0011	RESERVED
0100	RESERVED
0101	RESERVED
0110	RESERVED
0111	RESERVED
1000	RESERVED
1001	RESERVED
1010	RESERVED
1011	RESERVED
1100	RESERVED
1101	RESERVED
1110	RESERVED
1111	INTERRUPT ACKNOWLEDGE

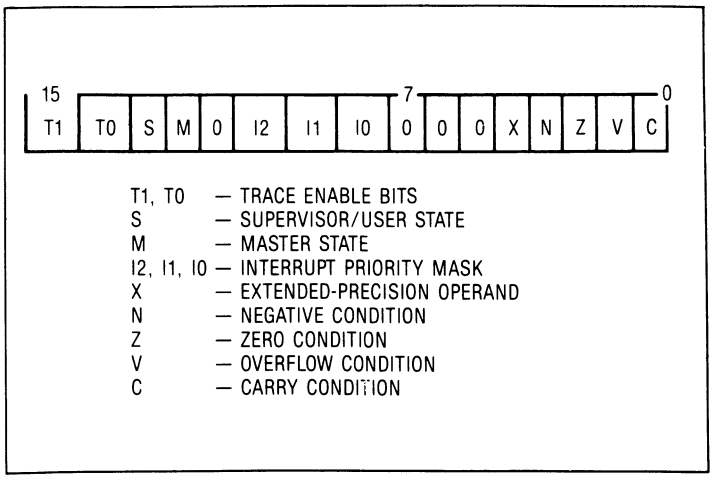

Figure 3. MC68020 status register.

and restore type are used for operating system context-switching support. These two types allow the saving and restoring of the state of a coprocessor in a virtual environment.

Once a coprocessor instruction has been passed from the main processor to the coprocessor for evaluation, the coprocessor has a number of main-processor-recognized primitives from which to implement a coprocessor instruction. The main processor supports 18 basic primitives, some with several variations. Primitives may be grouped into five basic categories. These involve processor/coprocessor synchronization, instruction stream manipulation, exception handling, general operand transfer, and register transfer.

Processor/coprocessor synchronization primitives allow the coprocessor to inform the main processor of its status. The coprocessor may signal that it is busy with the previous or current instruction. The coprocessor may allow the main processor to process any pending interrupts while it is busy. It may request that the main processor pass the current program counter value. It may tell the main processor to proceed with the next instruction, depending on whether instruction tracing is enabled or disabled. Finally, it may indicate the result of a conditional test.

Instruction stream manipulation primitives allow the coprocessor to request the transfer of the coprocessor instruction (F-line) to itself. The transfer_instruction_ stream primitive allows the coprocessor to request the main processor to transfer up to 256 bytes from the instruction stream to itself and to update the program counter as appropriate. Thus, coprocessor instructions may be any number of words long.

Exception handling primitives force the main processor to check for and/or to take an exception. The coprocessor may request the main processor to perform a supervisor check prior to the execution of a privileged coprocessor instruction. If the main processor is not in the supervisor state, a privilege violation exception is taken. If the coprocessor detects an exception before, during, or after the execution of a coprocessor instruction, it may force the main processor to take an exception and to save enough state information to recover from the exception later.

General operand transfer primitives are used to request the transfer of operands from or to the coprocessor. These primitives include evaluating an effective address for the coprocessor, transferring the evaluated effective address to the coprocessor, transferring data from/to the effective address, and transferring data to a previously evaluated effective address. In addition, the coprocessor may pass the main processor an address and request operand transfer to/from the address passed. An additional primitive allows transfer of operands to/from the top of the active system stack.

Register operand transfer primitives allow transfer of single main processor address/data registers, single main processor control registers, multiple main processor ad-

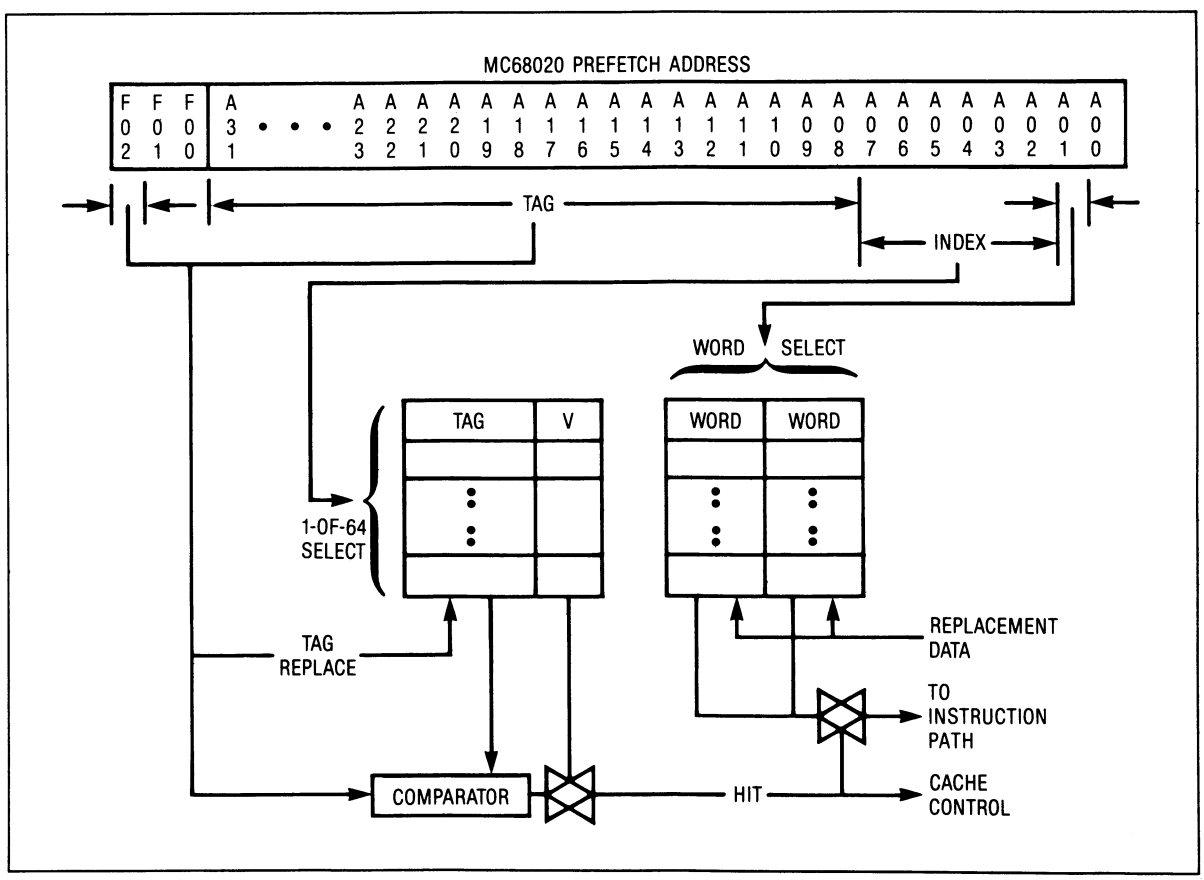

Figure 4. MC68020 on-chip cache.

dress/data registers, and multiple coprocessor registers between the main processor and the coprocessor. In addition, the coprocessor may request a transfer from/to the main processor status register and program counter. This allows the coprocessor to modify the processing state and processing flow of the main processor, which can act as a mechanism for task control.

Rather than attempting to define a complete set of macro-level special-purpose instructions, the MC68020 provides the general coprocessor interface to allow the system designer the freedom to enhance the main processor's instruction set.

On-chip instruction cache

Caches have been used for years in large machines to increase performance without greatly increasing system cost and complexity. It is possible to design a cache that is only a fraction of the size of the main store and that yet provides a significant decrease in the average access time to the main store. This concept has been incorporated into the MC68020—it contains a 256-byte-on-chip instruction cache which is used to obtain a significant increase in performance by reducing the number of fetches required to external memory. The reduced bus utilization by the MC68020 also increases system performance by providing more bus bandwidth for other bus masters such as DMA devices. The MC68020 only caches instruction stream accesses, to avoid stale data and other data-cache-related problems. The MC68020's cache configuration is shown in Figure 4. The cache interface to the processor data paths allows complete overlap of instruction fetches with data operand accesses, and thus provides a significant increase in performance. If simultaneous instruction and data operand requests are generated by the micromachine, a hit in the instruction cache allows concurrent fetches to take place.

A cache control register is provided to allow the operating system to maintain and optimize the cache. This register is shown in Figure 5.

The enable bit controls the MC68020's use of the cache. If this bit is set, the cache is enabled; if it is clear, the cache is disabled and the processor always fetches from external memory. Writing a one to the clear bit clears all valid bits in the cache and thus forces the MC68020 to refill the cache with new data. The freeze bit, when set, will not allow the MC68020 to replace valid data in the cache. The clear entry bit works in conjunction with an address in the cache address register. When a one is written to the clear entry bit, the location in the cache corresponding to the address in the cache address register is invalidated.

High-speed asynchronous bus

The MC68020 maintains the fast, easy-to-use, nonmultiplexed asynchronous bus interface of the M68000 family. However, several significant new features have been designed into the MC68020 bus interface. Figure 6 shows the MC68020's bus interface structure.

The bus interface consists of full 32-bit address and data buses, sequence control lines, and miscellaneous lines for interrupt and arbitration control. The 32-bit address and data buses are nonmultiplexed for maximum performance and simple interface design. This and other improvements allow the interface to support a 180-ns bus cycle time at 16.67 MHz, with an address-valid-to-data-valid specification of 120 ns. A typical bus cycle is shown in Figure 7.

Dynamic bus sizing. The MC68020 bus interface supports an important new feature known as dynamic bus sizing. This feature allows the MC68020 to "size" the bus port on a cycle-by-cycle basis, and to transfer data to and from 8-, 16-, and 32-bit bus ports. The key elements used to implement this new concept are the data multiplexer, the SIZE outputs, and the DSACKx inputs, which the MC68020 uses to dynamically interface to the various-sized buses on a bus-cycle-by-bus-cycle basis. Dynamic bus sizing allows software design to ignore the hardware configuration of memory and I/O ports and view the address space as a contiguous block of any granularity, be it byte, word, or long word. In addition to providing a very powerful and flexible system, it gives the user a high-performance processor that can run in an 8- or 16-bit system, or in a system configured with various-sized buses to individual elements. Figure 8 shows the various port configurations possible with the MC68020, while Figure 9 shows a typical operand transfer in which dynamic sizing is used to transfer long-word operands to 8- and 16-bit buses. As the MC68020 steps through memory, the two size lines (SIZ1, SIZ0) indicate the amount of data available for each transfer (00 = long-word, 01 = byte, 10 = word, and 11 = three-

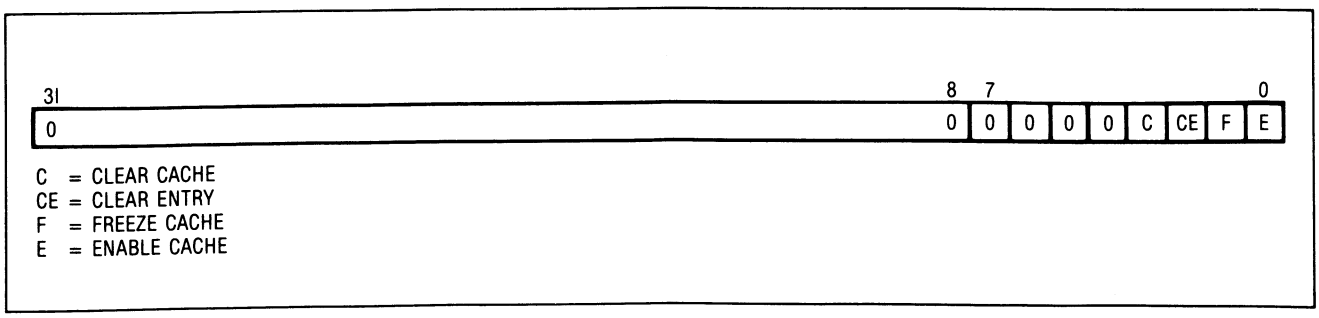

Figure 5. MC68020 cache control register.

byte). For each example in Figure 9, the memory control responds on the DSACKx pins (DSACK0, DSACK1) to define the port size, and is used by the processor as a factor for subsequent bus cycles, as needed to completely transfer the data. Since the MC68020 is able to dynamically size the bus, system configurations consisting of a mixture of 8-, 16-, and 32-bit memory cards and/or I/O ports are easily supported. Thus, the extension to a 32-bit external bus does not render older peripheral and memory cards obsolete.

In addition to providing the dynamic sizing feature, the MC68020 places no restrictions on operand alignment in memory; that is, data operands may appear on odd-byte boundaries and the processor will correctly fetch and store them.

Testability

There are three roughly equal components that contribute to the cost of a complex microprocessor: die cost, package cost, and assembly and test costs. Recognizing that the complexity of the MC68020 had increased significantly over previous members of the processor family, the design group decided that the development

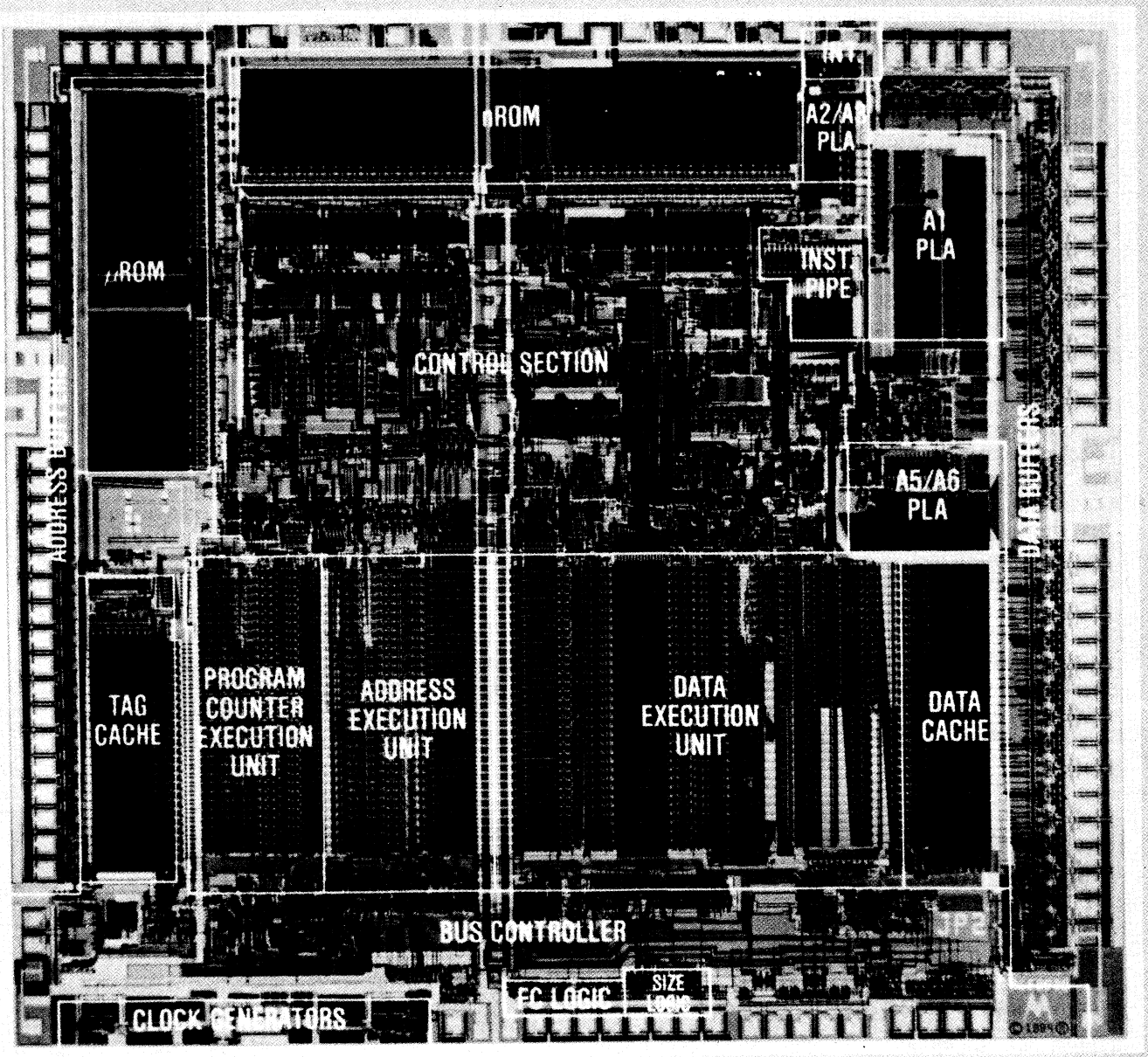

Figure 1. Die photograph of the MC68020.

of methods for assisting in production testing was necessary. Special test circuitry was therefore incorporated into the MC68020. It is designed for use only in a specialized test environment, where it can be used to exercise portions of the chip which would be difficult to test otherwise. Due to its specialized nature, the test circuitry is not intended to be exercised by the end user of the product.

The test circuitry enables various portions of the chip to be verified for functionality. The microROM and nanoROM contents can be multiplexed onto the external pins so that the control store of the micromachine can be verified without having to run large numbers of instructions to exercise every control word. The decode PLAs (A1 and A5/A6) have microcode sequences which decode all possible instruction words and accumulate the resulting outputs into a signature register that can then be read out and checked. The secondary decode PLAs (A2) also have microcode sequences which exercise them and write the results out. Finally, there is special microcode to support RAM tests on the instruction cache.

The test support provides controlled access to the structures of the chip that are the most difficult to test, thus reducing test costs by decreasing the number of test se-

The MC68020 is a high-performance 32-bit microprocessor. It is the first microprocessor to have evolved from a 16-bit machine to a full 32-bit machine that provides 32-bit address and data buses as well as 32-bit internal structures. Many techniques were utilized to improve performance and at the same time maintain compatibility with the other processors of the M68000 family. Among the improvements are new addressing modes which better support high-level-language structures, an expanded instruction set which provides 32-bit operations for the limited cases not supported by the MC68000 and the MC68010, and several new instructions which support new data types and provide solutions to special-purpose application problems. For those special-purpose applications for which a general-purpose processor alone is not adequate, a coprocessor interface is also provided.

Physical characteristics. The MC68020 is 9.22 mm on a side. It was designed with an HCMOS technology to minimize power consumption (it consumes 1 to 1.5 watts) and allow for high clock frequencies (16 MHz, for the worst case). It utilizes about 190,000 transistors, 103,000 of which are actually implemented. The package is a 114 PGA type (13 pins on a side with a depopulated center).

Figure 1 is an annotated photo of the MC68020 die.

Functional blocks. Figure 2 is a block diagram of the MC68020. The processor can be divided into three main sections: the bus controller, the micromachine, and the miscellaneous area. This division reflects the autonomy with which the sections operate. The bus controller comprises the address and data pads and multiplexers required to support dynamic bus sizing, a macro bus controller which schedules the bus cycles on the basis of priority, two micro bus controllers—one to control the bus cycles for operand accesses and the other to control the bus cycles for instruction accesses, and the instruction cache, with its associated control. The micromachine consists of an execution unit, ROM control stores, decode PLAs, an instruction pipe, and miscellaneous control sections. The execution unit has three sections: the instruction address section, the operand address section, and the data section. Microcode control is provided by a modified two-level store of microrom and nanorom. Decode PLAs are used to provide sequencing information. The instruction pipe and other miscellaneous control sections provide the secondary decode of instructions and generate the actual control signals that result in the decoding and interpretation of the control store.

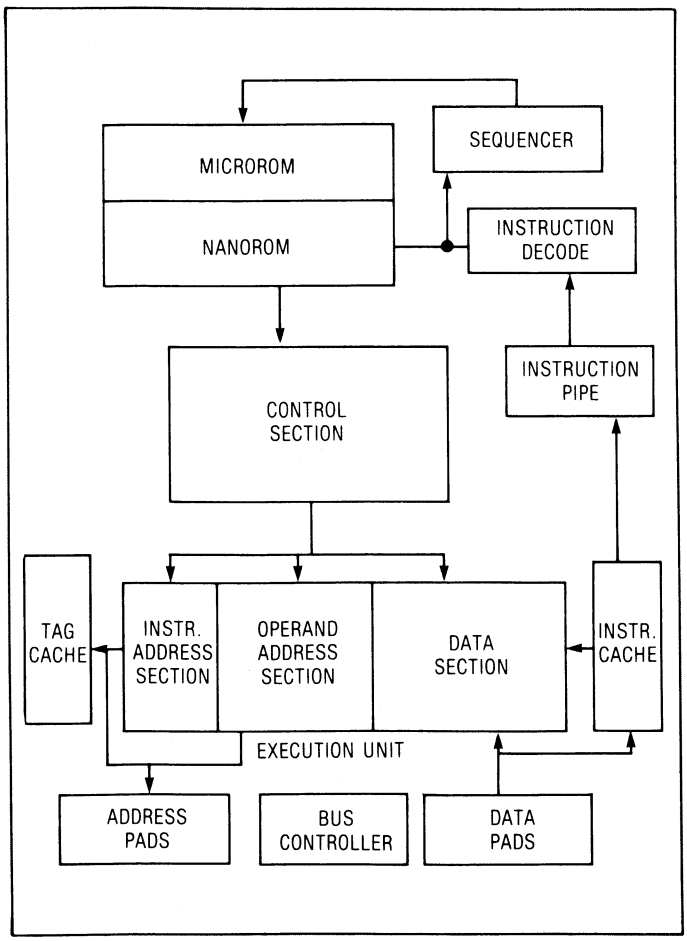

Figure 2. Block diagram of the MC68020.

quences which would otherwise be necessary. The chip area dedicated to actual test hardware and firmware is minimal with respect to the overall die size.

Performance

The primary design goals of the MC68020 were compatibility and performance. The chip's performance was significantly increased through the use of many different techniques—parallelism, pipelining, increased clock frequency, increased bus width, special-purpose units, new instructions and addressing modes, and an instruction cache.

Increased clock frequency. One of the simplest ways to increase performance is to increase the clock frequency at which a part executes. The MC68000 and its derivative, the MC68010, have a frequency range of 8 MHz to 12.5 MHz. On the MC68020, the base frequency has been increased to 16 MHz, and it is is not unreasonable to expect 20 MHz or above in the future. Though the basic clock frequency is roughly doubled, performance is not correspondingly doubled, since there are dependencies on external factors such as memory access times. Performance plots (see pages 116-118) illustrate the effect of increased clock frequencies over a range of memory access times. It is possible to see a compression of performance between clock speeds as the number of wait states increase.

Increased bus width. Another significant and relatively easy method for increasing performance is to increase the width of the data bus from 16 bits to 32 bits. This lets a single 32-bit access replace the two 16-bit accesses required to fetch a long operand. This reduction in bus accesses applies for both instruction and data accesses. An estimate of the performance improvement associated with the wider bus can be obtained from observing the performance difference between the MC68008 (8-bit data bus) and the MC68000 (16-bit data bus). Except for the difference in bus width, the MC68000 is identical to the MC68008. The MC68000 provides 1.66 times the performance of the MC68008. With the MC68020 it is not reasonable to expect a performance improvement this large, since the percentage of operand accesses which are long (and thus speeded up on the 16-to-32-bit increase) is lower than the percentage of word and long-word data

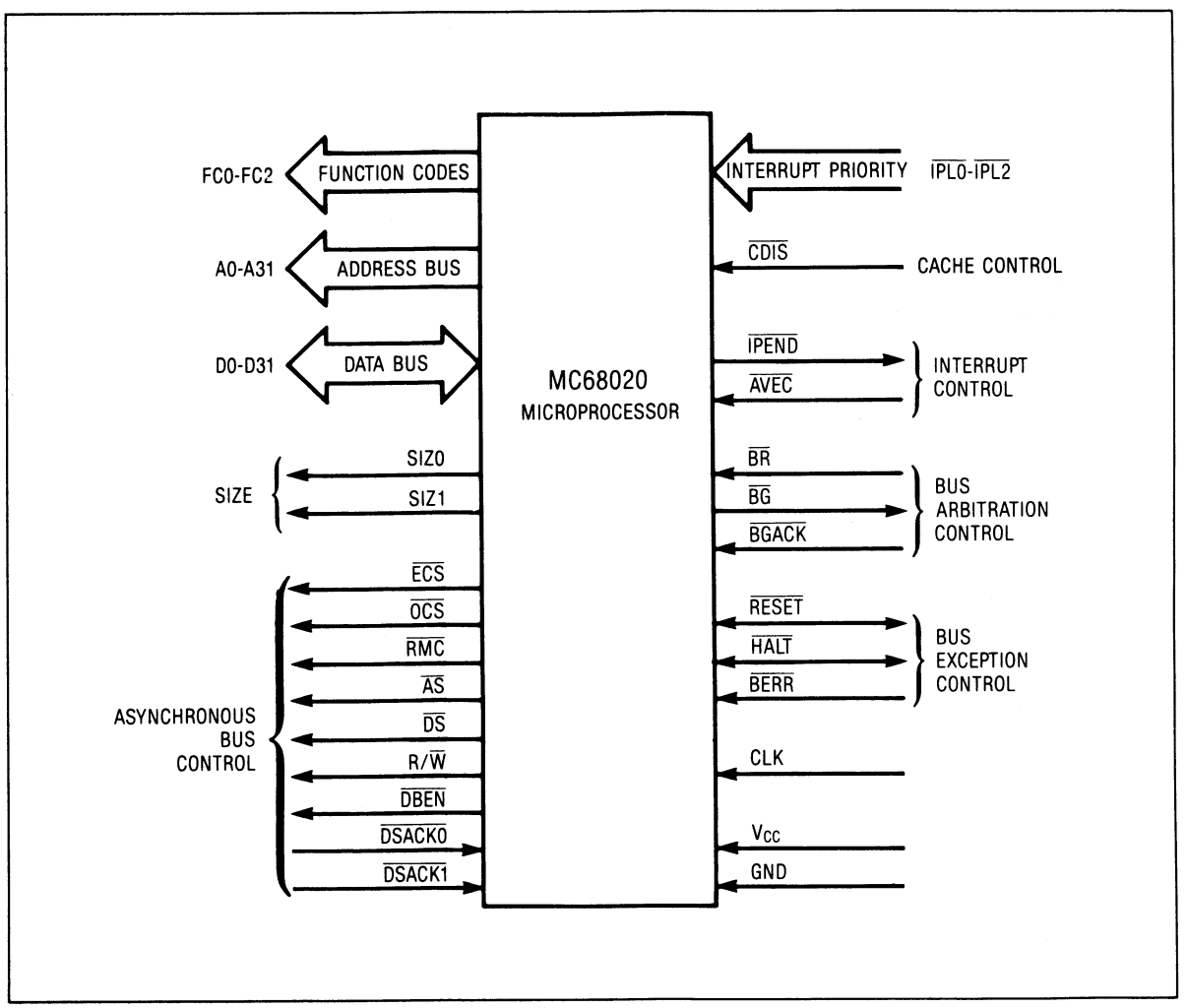

Figure 6. MC68020 bus interface.

accesses (which are improved by the 8-to-16-bit increase). However, there is an appreciable gain from the 32-bit bus for instruction accesses. Although instructions are of word length and are fetched a word at a time in the M68000, there is a benefit in fetching two words at a time, because the instruction stream is sequential in nature. The MC68020 stores the second word in a temporary register. This implies that even with the cache disabled or while executing in-line code, the MC68020 will realize a reduction in the number of bus cycles approaching 50 percent.

Instruction cache. The goal when designing a microprocessor such as the MC68020 is to build a fast, well-balanced machine. It is now possible to build a machine that is very fast internally, but that machine must still communicate with the external world. Thus it is important to minimize the delay due to executing bus cycles. This is done by reducing the minimum bus cycle time from four clock cycles to three clock cycles, and by reducing the number of bus cycles to be excuted by providing a 32-bit data bus and a 256-byte instruction cache. The instruction cache reduces the execution time in two ways. First, it provides a two-clock-cycle access time for an access that hits in the cache, and second, if the access hits in the cache, it allows simultaneous instruction and data accesses to occur. Of these two benefits, simultaneous instruction and data access is more significant since it allows a 100 percent reduction in the time required to access the instruction rather than the 33 percent reduction afforded by going from three clocks to two clocks. Reducing cache access time further (below two clocks) would not be beneficial, since the time required to update the instruction stream pointers would be two clocks as well. To understand how it is possible to completely hide the time needed to access an instruction, it is necessary to observe the parallelism that exists between the instruction fetch mechanism and the data fetch mechanism.

Figure 10 illustrates that data and instruction addresses are calculated in parallel and have separate paths to the address pads. This allows a simultaneous instruction and data access if there is a hit in the cache while a data access is taking place.

Another advantage of the separated instruction and data address bus is that it allows for pipelined accesses within the cache. Because of the sequential nature of instruction accesses, it is easy to determine the next-in-line instruction address. There is normally no performance penalty in assuming that the access will need to take place in the next microinstruction and then starting the access a clock early, as long as this does not require the external bus. Using this strategy, one can design a system in which there is no penalty for detecting a miss in the cache. When a miss is detected, the instruction address is at the pads ready to run a bus cycle. There is also a special mechanism available to bias the address pads toward data when there is a penalty for the normal instruction bias.

Pipelining. The micromachine of the MC68020 is highly pipelined. The predominant pipeline mechanism is the instruction pipe, although there are several other pipelined mechanisms to provide control to various units within the micromachine. It is important to define the terms used here, for too often the term *instruction pipe* is used to describe an *instruction queue.* The distinction is this—when a change of instruction flow occurs (i.e., a branch), the instruction pipe must be filled before execution of the new instruction can occur, whereas the instruction queue must have only the first word present. However, an instruction queue does not provide the performance benefits of a pipe during the execution of in-line code. Instruction queues tend to saturate the bus with instruction fetches that are discarded following a change-of-flow instruction. When one considers that a change-in-flow instruction occurs 25 to 30 percent of the time, and that

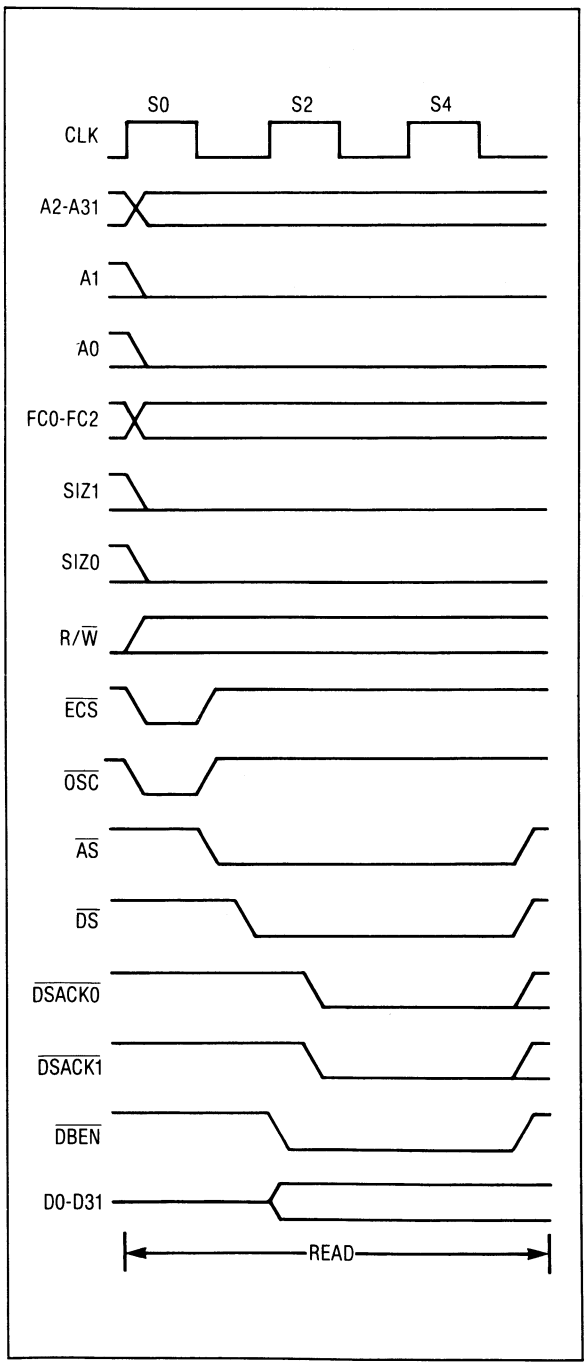

Figure 7. Typical MC68020 bus cycle.

those instruction accesses that may be discarded can prevent necessary data accesses, the advantage of an instruction pipe is evident.

The depth of the instruction pipe on the MC68020 is three words. While there are advantages to increasing the depth of the instruction pipe, there are also disadvantages (or at least many situations in which the advantages are obviated) if the depth of the pipe is not gauged correctly. It is necessary to balance the depth of the pipe against the performance of the micromachine. The advantage of an instruction pipe is in staging the activity associated with instruction execution in such a way that the time required to decode the instruction and to provide the control necessary to execute it is not visible outside the chip. This is done by executing a different instruction in each of the three stages simultaneously. Figure 11 shows the stages of execution for the three-word instruction pipe.

The depth of the instruction pipe was increased from two to three words on the MC68020, since it was observed that on a 32-bit bus there is no performance penalty for the third element during a change-of-flow instruction. This is true because it is always possible to fetch three words in two accesses. Because of the possibility of branching to an odd-word address, in which case two accesses are required to fetch either two or three words, the minimum number of accesses required to branch is two. A demonstration of the stubborn consistency of this principle is given by Figure 12, which depicts two memory configurations for the instruction stream A, B, C, D, E. If a branch is made to the first example (even aligned), only one access is required to fetch two words. However, in the other case (odd aligned), two accesses must occur to fetch the first two words. Once it is determined that two accesses need to be made to fill the pipe, it can be observed that with those two accesses it is always possible to fetch the first three words. Thus there is no penalty for increasing the depth of the instruction pipe from two to three words.

As described above, the greater depth of the instruction pipe, while improving performance of in-line code, has a detrimental effect on change-of-flow instructions. By using a three-word pipe, the MC68020 optimizes the performance of the micromachine.

Parallelism. Parallelism is the concurrent execution of several tasks. Perhaps the most significant case of parallel operations in the MC68020 can be found in the execution unit (see Figure 13). The execution unit is divided into three 32-bit sections, each with a 32-bit adder: the instruction address section, in which the instruction addresses are calculated and the pointers are stored; the operand address section, in which the operand addresses are calculated; and the data section, in which the data operations can take place. Although each section is optimized to perform its primary function, each can function as a general computation unit, when needed, and can

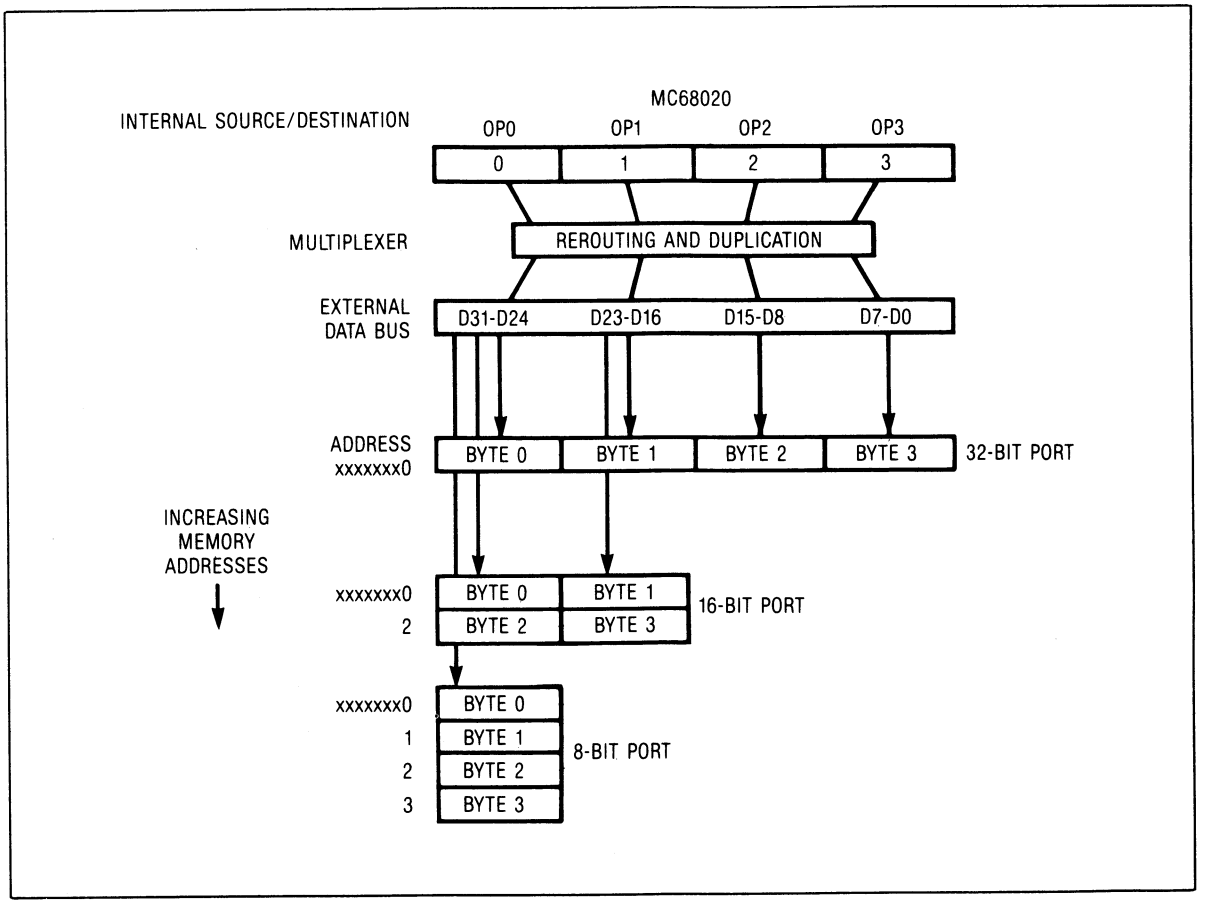

Figure 8. MC68020 data bus interface to various-sized ports.

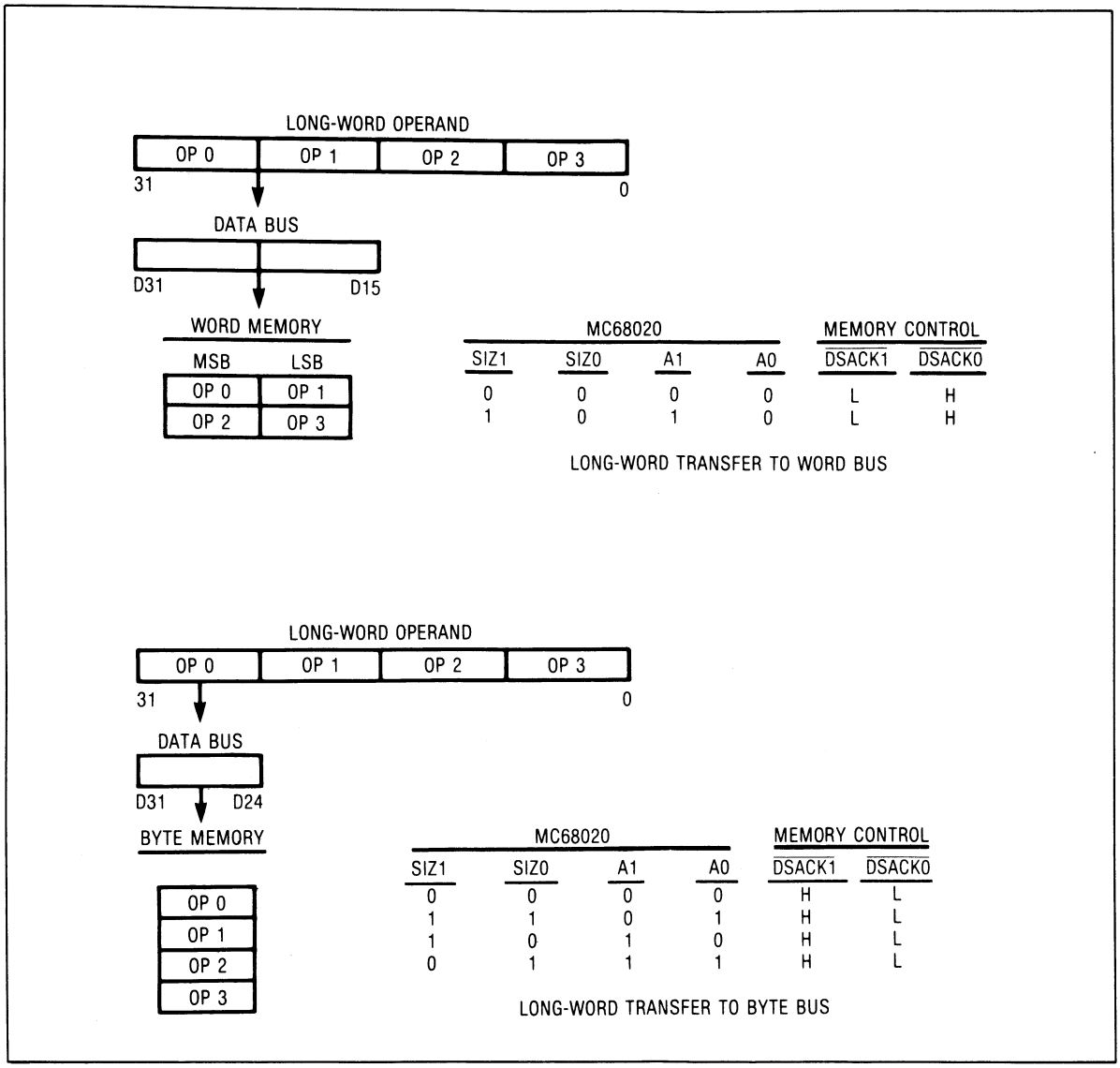

Figure 9. MC68020 dynamic bus sizing.

be connected to its neighbor via a high-speed bus. For example, a 32 × 32 multiply producing a 64-bit product utilizes both the operand address section and the data section.

The division of instruction address generation and operand address generation is carried beyond the execution unit (see again Figure 10), allowing simultaneous instruction and data accesses. This is another case of parallel operation.

There are two semiautonomous machines in the MC68020—the bus controller, which schedules and controls all bus activity, and the micromachine, which controls the execution of the instruction by requesting bus activity and controlling the execution unit and instruction pipe. There is a significant amount of parallelism between the micromachine and the bus controller. When an instruction has completed execution within the execution unit, a write of the results of memory may still be pending. If this is the case, there is no reason why the micromachine cannot proceed with the execution of subsequent instructions, as long as they do not require the bus resources needed to complete the write. This is indeed what happens in the example shown in Figure 14,

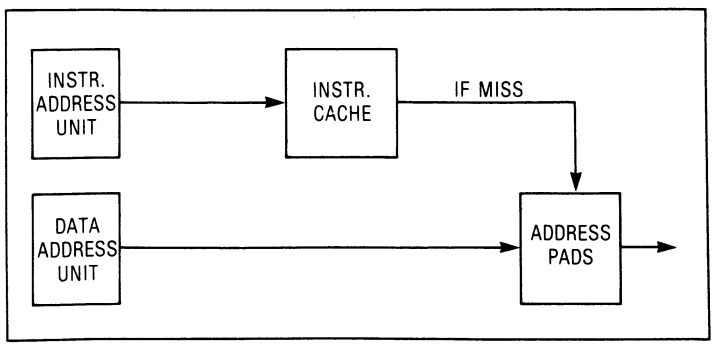

Figure 10. Parallel calculation of instruction and data addresses.

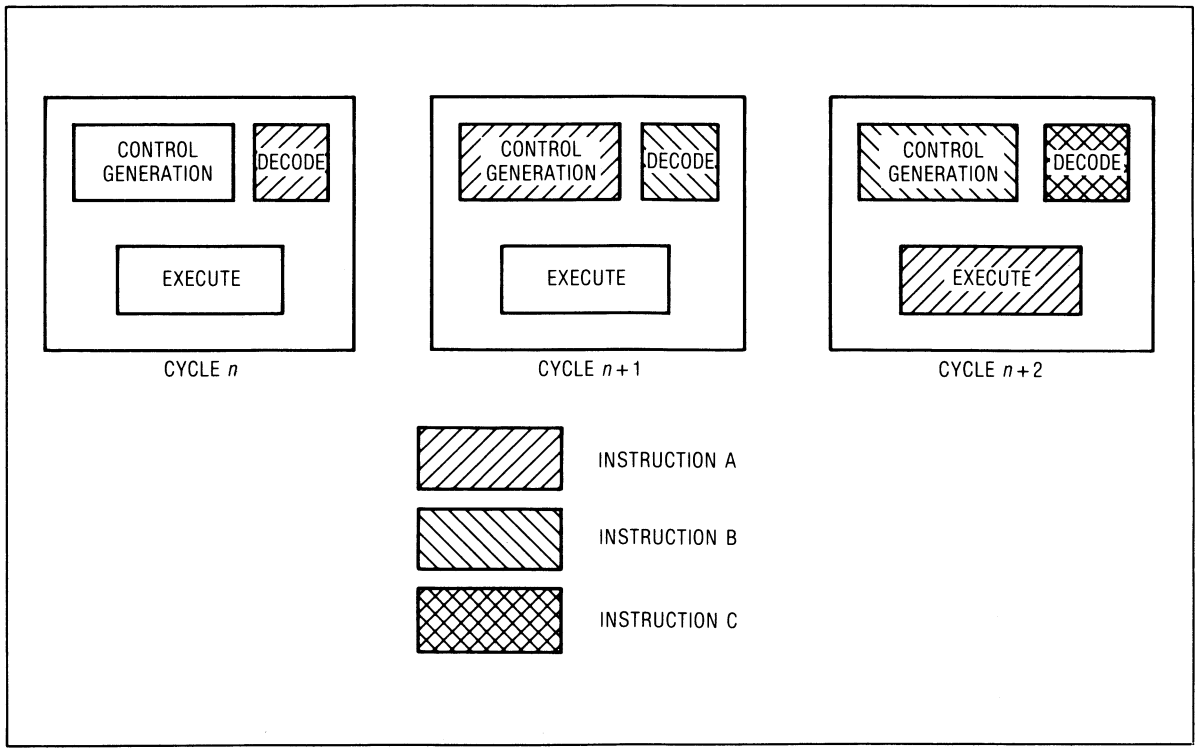

Figure 11. Stages of execution for the three-word instruction pipe.

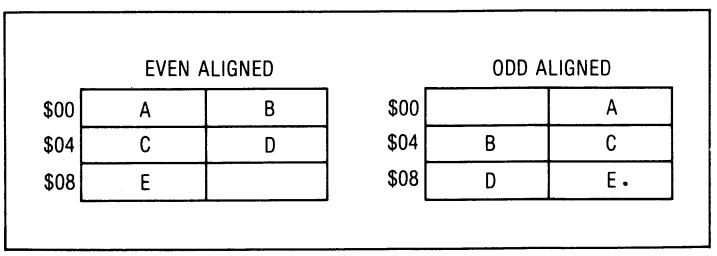

Figure 12. Two memory configurations for the instruction stream A, B, C, D, E.

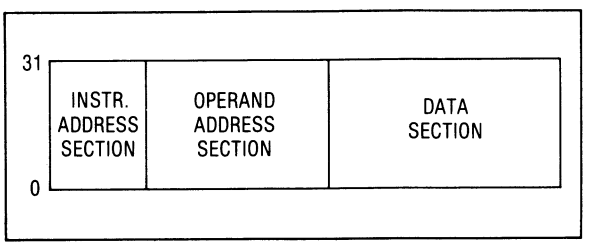

Figure 13. MC68020 partitioned execution unit.

in which two instructions are executed with the cache disabled, using a 32-bit bus with no wait states. In this case, the two instructions, MOVE to memory followed by ADD register to register, overlap. The write associated with the MOVE instruction occurs in parallel with the execution of the ADD. Because the MOVE instruction has completed all of its activity within the execution unit and there remains only bus controller activity, the micromachine proceeds with the execution of the ADD instruction. Because the ADD instruction does need to perform an instruction access to fill the pipe, the ability to perform simultaneous instruction and data accesses becomes more significant. This overlap of instructions can occur over any number of instructions and is solely dependent upon the length of the bus cycle and the need of the overlapped instructions to use the bus. If succeeding instructions have no need to use the external bus—as is the case for register-to-register operations, branching, and other instructions which hit in the instruction cache and have no external data transfers—then these instructions may have their execution times totally absorbed by the previous write cycle. In our example, the execution time of the ADD would appear to be zero clocks outside the chip. The overlap capability is particularly important for systems with an external cache. With a data cache, writes will typically require longer cycles because of the write-through problem. For example, a write may take eight or more clock cycles compared to reads, which the external cache can complete in three cycles. Providing overlap capability on these write cycles helps mitigate the system penalty associated with a data cache.

New instructions and addressing modes. The instruction set was extended to provide further high-level-language (HLL) support. The most significant aspect of these extensions are the additions that have been made to the addressing modes. The indexed addressing mode now makes it possible to

- scale indexes,
- use memory indirection,
- have 16-bit and 32-bit displacements,

- use preindex or postindex on memory indirect, and
- suppress any element of the address calculation.

The significance of these additions can be readily seen in the sample HLL statement shown in Figure 15. This particular example happens to be written in Pascal, but the performance improvement obtained here would be common to any subscripted operation in an HLL.

The figure shows the code for a Pascal statement compiled for both the MC68000 and the MC68020. The number of clocks and bus cycles assumes no wait state accesses; the cache is disabled for the MC68020. With the enhanced addressing modes, it is possible to replace three or more MC68000/010 instructions with a single MC68020 instruction. In the above case, two noticeable weaknesses of the MC68000 addressing modes are the inability to perform scaling on index values during the calculation and the lack of 16- or 32-bit displacements with the index addressing mode. The MC68020 provides these capabilities, with significant performance results. This is particularly true for the case of scaling, since it adds no time to the effective address calculation time.

In Table 6, the effect that these modes have on performance for this line of code is presented. Although it is difficult to compare the machines exactly (e.g., the MC68000 requires four-cycle accesses whereas the MC68020 can use three-cycle accesses), the differences do not prevent an approximate evaluation.

The other additions to the instruction set fall into two categories—the extension of operand size and displacement size, and the addition of new complex instructions. The extension of operand and displacement size to a full 32 bits did not affect many instructions, since most instructions already dealt with 8-, 16-, and 32-bit operands. These new complex instructions, though they should contribute to performance improvements for general-purpose applications, will have their most profound effect on special-purpose applications. For example, the bit field instructions will be used throughout many applications but will be most significant for graphics and communications applications.

The coprocessor interface was defined to provide support for instruction set extensions. It provides a mechanism by which special-purpose units can be added to the system to directly attack specific problems. The MC68881 floating-point coprocessor is an example.

Special-purpose hardware. Special-purpose hardware was designed to enhance the computational power of the basic micromachine. This was done to provide faster execution for those instructions which require services beyond the operations performed by an ALU. A barrel shifter was added to the micromachine to support bit field instructions; it is also used for a variety of other instructions. Special control features were added to support multiplication of two bits per microinstruction. A significant addition was a scaling unit which, during the evaluation of an effective address, allows the scaling of index values without any delays.

Performance evaluation. The result of all these efforts is that the MC68020 can execute instructions very fast.

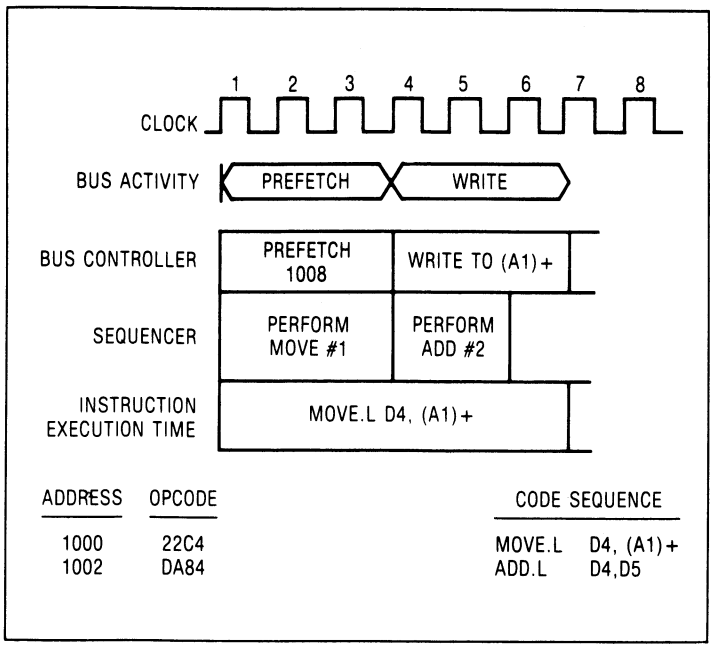

Figure 14. An example of processor overlap—two instructions are executed with the cache disabled, using a 32-bit bus with no wait states.

```
if piececount[class[i]] <> 0 then

Compiled 68000 code   (13-bytes)

 68000     68020
CLK  BC   CLK  BC
 4    1    3    1      move.w   d1,d3
 8    1    4    0      lsl.w    #1,d3
12    2    7    1      lea      db__register(db,d3.w),a2
12    3    8    2      move.w   ~class(a2),d3
 8    1    4    1      lsl.w    #1,d3
12    2    7    1      lea      db__register(db,d3.w),a2
12    3    8    2      tst.w    ~piececount(a2)
11    2    9    2      beq      else
79   15   50   10

Compiled 68020 code   (8-bytes)

          68020
         CLK  BC
          11   3      move.w   (~class,db,d1.w*2),d3
          12   2      tst.w    (~piececount,db,d3.w*2)
           9   2      beq      else
          32   7

CLK = clocks
BC  = bus cycles

d1  = i        a2 = temp ptr        db__register = 0
d3  = temp     db = base ptr        ~class       = $1c26
                                    ~piececount  = $1c40
```

Figure 15. Code for a Pascal statement compiled for both the MC68000 and the MC68020.

Table 6.
Execution characteristics for the Pascal statement.

	CLOCKS	EXECUTION TIME	PERFORM X 68000
EXECUTE 68000 CODE			
8-MHz 68000	79	9.88 μs	1
16-MHz 86020 (CACHE DISABLED)	50	3.16 μs	3.2
16-MHz 68020 (HIT IN THE CACHE)	42	2.63 μs	3.8
EXECUTE 68020 CODE			
16-MHz 68020 (CACHE DISABLED)	32	2.0 μs	4.9
16-MHz 68020 (HIT IN THE CACHE)	24	1.5 μs	6.6

NUMBER OF BYTES, 68000:68020 = 1:0.62

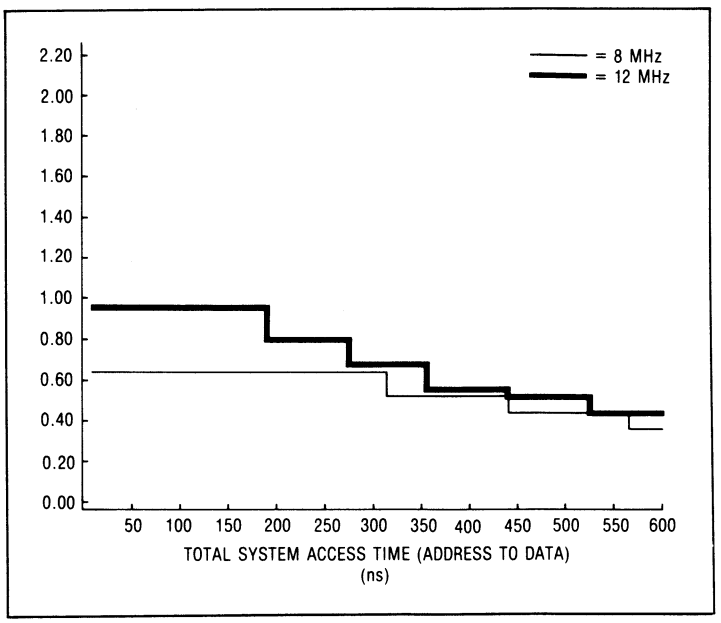

Figure 16. Performance analysis of the MC68010 at 8 MHz and 12 MHz.

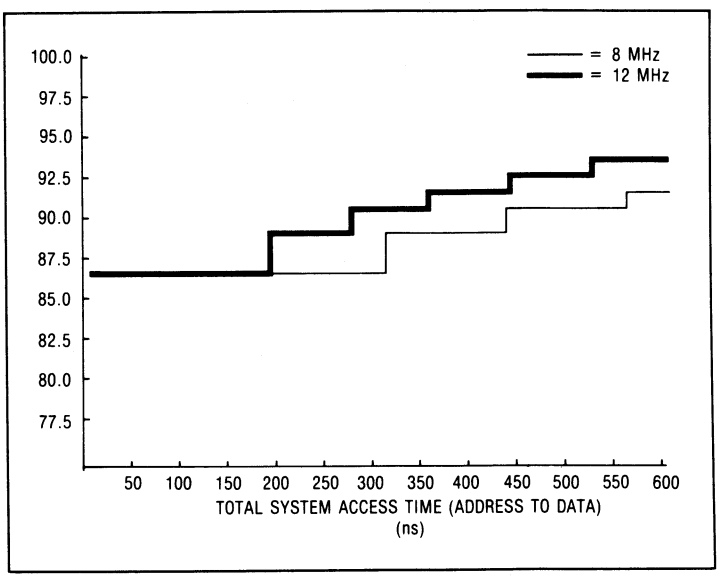

Figure 17. Bus utilization analysis of the MC68010 at 8 MHz and 12 MHz.

The bus is faster, the number of bus cycles has been reduced, and the micromachine is faster and more powerful. Hence, the average number of clocks needed to execute an instruction has dropped from about 13.5 in the MC68000 to about 7 in the MC68020, even though these clocks have at least half the period of those of the 68000 and the instructions are more complex.

It is difficult to evaluate the performance of a processor without drawing criticism. Benchmarks have been developed to provide standards against which to measure performance, but defining a benchmark only creates more controversy as to its validity. As difficult as it may be to evaluate performance, it is a necessary activity.

So, in order to measure the performance of the MC68020, we used a series of benchmarks. Since the MC68020 is a general-purpose processor, there is no single application program that will provide an adequate measure of performance. Therefore, we used an average of 18 equally weighted programs. These benchmarks were selected from four categories: traditional benchmarks, program development programs, design applications, and utilities. All programs were written in an HLL and compiled with an Apollo compiler.

The traditional benchmarks were Ackerman's Function, Puzzle, Prime, and the Towers of Hanoi, written in Pascal, C, and Fortran. The program development programs consisted of a C, Pascal, and Fortran compiler, a binder, and an M68000 assembler. The utility programs included a copy file, a compare file, a sort file, and a search for element in a file (grep). The application programs comprised a microcode assembler, a program to expand PLA terms, a text formatting program (nroff), and the performance evaluation program itself. These 18 benchmarks were selected to provide a group of very different programs, written in different languages, compiled by an equivalent compiler, and representing a good sampling of the computing that is done during the design of a microprocessor.

The programs evaluated, the number of instructions executed, and the results of the evaluations are listed in Table 7. There were three configurations measured for each program and two values obtained from each configuration. Measurements were taken for an 8-MHz MC68010, a 16-MHz MC68020 with the cache disabled, and a 16-MHz MC68020 in which all instruction accesses hit in the cache. For each of these configurations, the performance—as measured in millions of instructions per

Table 7.
Programs evaluated and the results of the evaluations.

	INSTRUCTIONS EXECUTED	68010		68020 NO CACHE		68020 IN CACHE	
		MIPS	BUS	MIPS	BUS	MIPS	BUS
BENCHMARKS							
1. ACKERMAN	3,100,203	0.7	93%	2.2	85%	2.8	66%
2. HANOI	32,505,848	0.6	94%	2.1	86%	2.6	74%
3. PRIME	1,978,626	0.7	90%	2.1	82%	2.6	63%
4. PUZZLE	8,190,412	0.7	72%	2.4	79%	2.9	60%
UTILITIES							
5. COMPARE FILE	4,968,773	0.7	86%	1.9	80%	2.4	60%
6. COPY FILE	180,825	0.5	89%	1.8	85%	2.3	71%
7. LIST DIRECTORY	494,179	0.6	80%	2.0	80%	2.5	64%
8. SORT FILE	50,170,012	0.8	84%	2.0	79%	2.5	63%
9. SEARCH ELEMENT (GREP)	14,269.122	0.6	95%	2.2	84%	2.7	67%
PROGRAM DEVELOPMENT							
10. C COMPILER	14,269,122	0.6	89%	2.2	83%	2.8	65%
11. PASCAL COMPILER	4,257,764	0.6	88%	2.2	83%	2.8	64%
12. FORTRAN COMPILER	597,926	0.6	86%	2.2	82%	2.8	64%
13. BINDER	65,667	0.6	90%	2.2	83%	2.8	67%
14. 68020 ASSEMBLER	13,936,569	0.6	90%	2.1	83%	2.6	66%
APPLICATIONS							
15. TEXT FORMAT (NROFF)	96,147,880	0.6	92%	2.2	84%	2.7	66%
16. MICROCODE ASSEMBLER	189,918,880	0.7	76%	2.5	79%	3.1	59%
17. PLA EXPANSION	109,748,040	0.6	92%	2.2	84%	2.7	68%
18. PERFORMANCE EVAL.	37,792,280	0.5	78%	1.8	79%	2.2	63%
TOTAL/AVERAGE	582,592,128	0.6	87%	2.1	82%	2.7	65%

second (MIPS) and as a percentage of bus utilization—was obtained. All measurements were based on no-wait-state memory accesses and, for the MC68020, on the use of a 32-bit data bus. The MIPS value was measured in MC68000 MIPS and was not standardized to any other machine. The measurements on the MC68020 were for the processor executing MC68000 code with no instruction overlap; thus, the MC68020 using the instruction and addressing mode enhancements was not measured, making the results conservative.

The data in Table 7 have been compiled and the average displayed in plot form (Figures 16-19). These plots indicate performance and bus utilization for the MC68010 and the MC68020. The performance measure (the *y* axis) is in MC68000 MIPS. The system access time (the *x* axis) provides a view of the effect of increasing the clock frequency of the processor for a particular system configuration. System access time includes address translation delays, buffer delays, bus delays, and memory access time. The *x*-axis calibration is in nanoseconds and represents the access time from address valid to data.

A comparison of two Puzzle benchmarks written in Pascal, one with and one without the new addressing modes, shows the performance improvement that can be realized by using those modes (see Table 8). In this comparison, the MC68020 and the MC68010 programs differ only in that scaling requires no time and 16-bit displacements are available with the index addressing mode. The table shows the number of instructions and the number of clocks required to execute the Puzzle program.

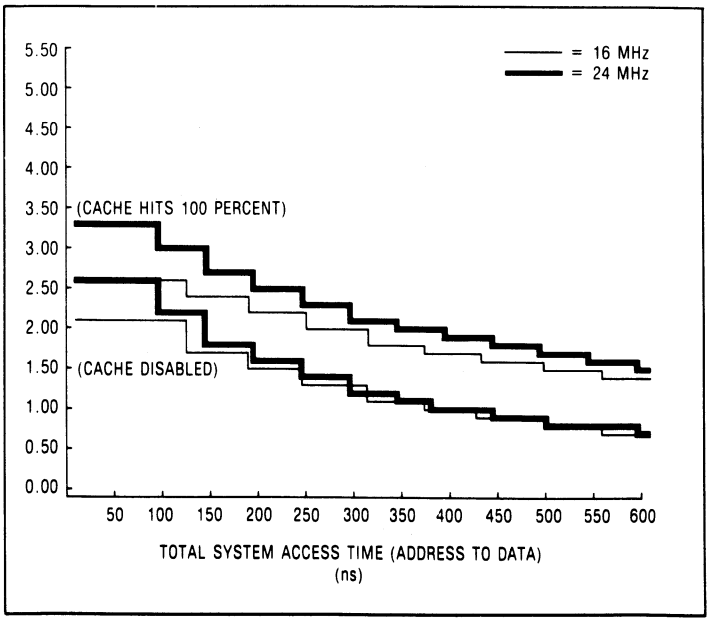

Figure 18. Performance analysis of the MC68020 at 16 MHz and 24 MHz.

On the performance plots, two performance levels are given for each processor to show the effect of increasing the clock frequency. For the MC68010, the clock frequencies are 8 MHz and 12 MHz; for the MC68020, the frequencies are 16 MHz and 24 MHz. Also for the 68020,

Table 8.
68000 to 68020 clocks/instruction comparison for the Puzzle benchmark.

EXECUTING 68000 INSTRUCTION SET
8,190,412 INSTRUCTIONS

PROCESSOR	CLOCKS	AVERAGE CLOCKS/INSTRUCTION
68010	93,295,280	11.4 CLOCKS/INSTRUCTION
68020 NO CACHE	54,003,768	6.6 CLOCKS/INSTRUCTION
68020 IN CACHE	44,664,960	5.5 CLOCKS/INSTRUCTION

EXECUTING 68020 INSTRUCTION SET
7,066,891 INSTRUCTIONS

PROCESSOR	CLOCKS	AVERAGE CLOCKS/INSTRUCTION
68020 NO CACHE	48,761,547	6.9 CLOCKS/INSTRUCTION
68020 IN CACHE	39,574,589	5.6 CLOCKS/INSTRUCTION

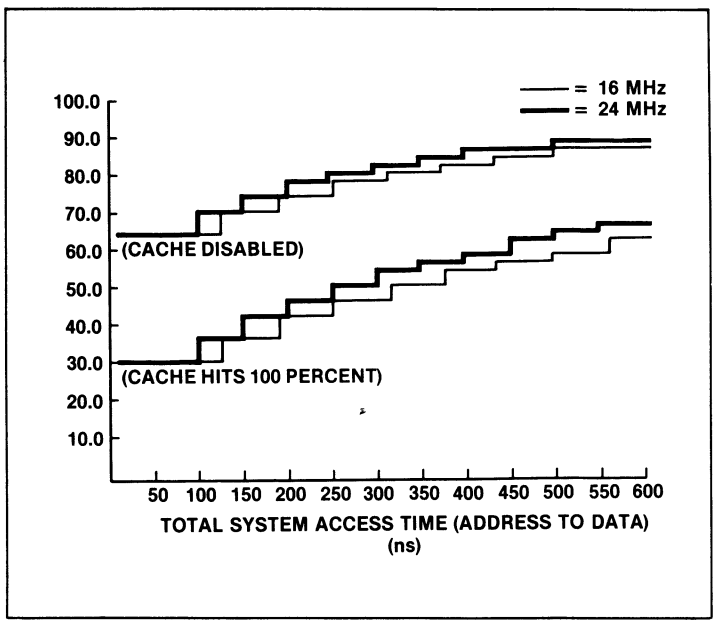

Figure 19. Bus utilization analysis of the MC68020 at 16 MHz and 24 MHz.

performance is represented by two sets of figures—one for performance with the cache disabled and one for performance with a 100-percent cache hit ratio. This makes it possible, given the nature of the program, to interpolate the performance level for a given cache hit rate.

The MC68020 represents a new 32-bit performance standard, we believe. Performance was not our only design criterion, however. We also maintained compatibility with the M68000 family, to protect the value of users' existing software, and yet provided enough instruction set enhancements to make the MC68020 useful in the widest possible range of applications. ■

The authors: MacGregor (left), Mothersole (center), and Moyer (right).

Doug MacGregor designed the control structures and wrote the microcode for the 68010 and the 68020. Aside from picking up a BA in history and Asian studies at night, he got some maturity and direction while serving in the Navy. Deciding that eating had some attractive merits, he attended the University of Illinois, where he obtained an MS in computer science and after which he began work at Motorola. He intends to begin work for his doctorate at the University of Kyoto at the end of this year. His passions include acoustic analysis of submarines, etymology, and efficiency.

Dave Mothersole was project leader for the MC68020 design. He has spent six years designing M68000 processors and is currently researching new VLSI computer architectures. His interests include computer networking, some form of basketball, and sailing. A member of IEEE, he holds a BS and MS in electrical engineering from the University of Texas at Austin.

Bill Moyer is a systems design engineer in the high-end microprocessor design organization at Motorola. He has been involved with the definition and implementation of a variety of processors in the M68000 family, including the MC68008, MC68010, and the MC68020. His areas of interest include control and sequencer logic design, bus architectures, and design of software tools for PLA development. He holds a BS in electrical engineering from Rice University and is currently completing his MSEE degree at the University of Texas.

Questions about this article can be directed to the authors at Motorola, Inc., Semiconductor Products Sector, MOS Microprocessor Products Group, 3501 Ed Bluestein Blvd., Austin, TX 78721.

MOTOROLA *Semiconductor Products Inc.*

3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721 • A SUBSIDIARY OF MOTOROLA INC.

A18629 PRINTED IN USA 2-85 IMPERIAL LITHO C28460 18,000

AR217